LendingClub

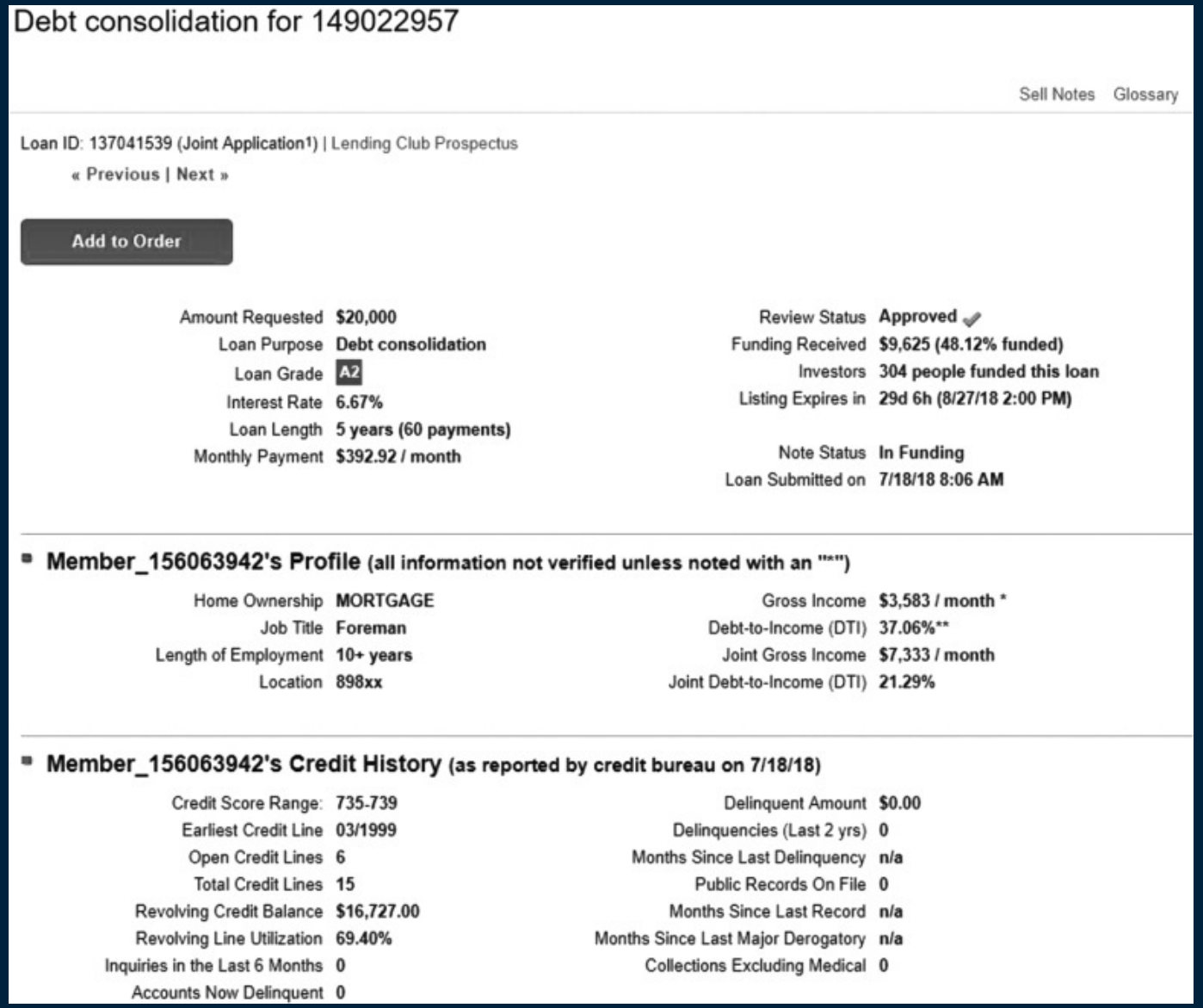
Predictive Models and Decision Trees

Lecture plan

- Feature engineering
- ID3 algorithm
- The sklearn library
- Representation of a decision tree
- Analyzing a decision tree
- Model independence
- Data pre-processing

Data available on each loan application

- **We think of these fields of information as attributes**



Debt consolidation for 149022957

Sell Notes Glossary

Loan ID: 137041539 (Joint Application1) | Lending Club Prospectus

« Previous | Next »

Add to Order

Amount Requested	$20,000	Review Status	Approved
Loan Purpose	Debt consolidation	Funding Received	$9,625 (48.12% funded)
Loan Grade	A2	Investors	304 people funded this loan
Interest Rate	6.67%	Listing Expires in	29d 6h (8/27/18 2:00 PM)
Loan Length	5 years (60 payments)		
Monthly Payment	$392.92 / month	Note Status	In Funding
		Loan Submitted on	7/18/18 8:06 AM

Member_156063942's Profile (all information not verified unless noted with an "*")

Home Ownership	MORTGAGE	Gross Income	$3,583 / month *
Job Title	Foreman	Debt-to-Income (DTI)	37.06%**
Length of Employment	10+ years	Joint Gross Income	$7,333 / month
Location	898xx	Joint Debt-to-Income (DTI)	21.29%

Member_156063942's Credit History (as reported by credit bureau on 7/18/18)

Credit Score Range:	735-739	Delinquent Amount	$0.00
Earliest Credit Line	03/1999	Delinquencies (Last 2 yrs)	0
Open Credit Lines	6	Months Since Last Delinquency	n/a
Total Credit Lines	15	Public Records On File	0
Revolving Credit Balance	$16,727.00	Months Since Last Record	n/a
Revolving Line Utilization	69.40%	Months Since Last Major Derogatory	n/a
Inquiries in the Last 6 Months	0	Collections Excluding Medical	0
Accounts Now Delinquent	0		

Attributes / Features

	id	loan_amnt	funded_amnt	grade	fico_range_high	fico_range_low	default
0	84044117	35000.0	35000.0	E	694.0	690.0	False
1	84393373	5000.0	5000.0	C	674.0	670.0	False
3	83645580	16000.0	16000.0	C	714.0	710.0	True
4	83983220	20000.0	20000.0	D	674.0	670.0	False
6	83904318	8000.0	8000.0	C	684.0	680.0	False
7	83902702	10000.0	10000.0	B	684.0	680.0	False
10	84040775	4500.0	4500.0	B	664.0	660.0	False
11	84101628	1500.0	1500.0	D	714.0	710.0	False
12	83598178	20000.0	20000.0	A	709.0	705.0	False
13	84333354	9000.0	9000.0	C	669.0	665.0	True

Attributes / Features

0 loan_amnt 8000.0
1 funded_amnt 8000.0
2 annual_inc 30000.0
3 dti 23.06
4 revol_bal 8220.0
5 delinq_2yrs 0.0
6 open_acc 9.0
7 pub_rec 0.0
8 fico_range_high 664.0
9 fico_range_low 660.0
10 revol_util 31.1
11 cr_hist 229.03139694860263
12 term:: 36 months 1.0
13 term::nan 0.0
14 verification_status::Not Verified 0.0
15 verification_status::Source Verified 1.0
16 verification_status::Verified 0.0
17 verification_status::nan 0.0
18 home_ownership::ANY 0.0
19 home_ownership::MORTGAGE 1.0
20 home_ownership::OWN 0.0
21 home_ownership::RENT 0.0
22 home_ownership::nan 0.0
23 emp_length::1 year 0.0
24 emp_length::10+ years 1.0
32 emp_length::9 years 0.0
33 emp_length::< 1 year 0.0
34 emp_length::nan 0.0
35 purpose::car 0.0
36 purpose::credit_card 0.0
37 purpose::debt_consolidation 1.0
38 purpose::home_improvement 0.0
39 purpose::house 0.0
40 purpose::major_purchase

Feature Engineering

- **Feature Engineering**: The process of using ***domain knowledge*** of the data to choose or create *attributes*/ *features* that make machine learning algorithms work
- **Feature templates**: the information required to create a feature: the layer where a feature will be stored, the attributes a feature is created with, and the default tool used to create that feature
- **Feature combination**: bringing together separate events to create a feature that is less trivial

> Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering.
>
> — *Andrew Ng, Machine Learning and AI via Brain simulations*

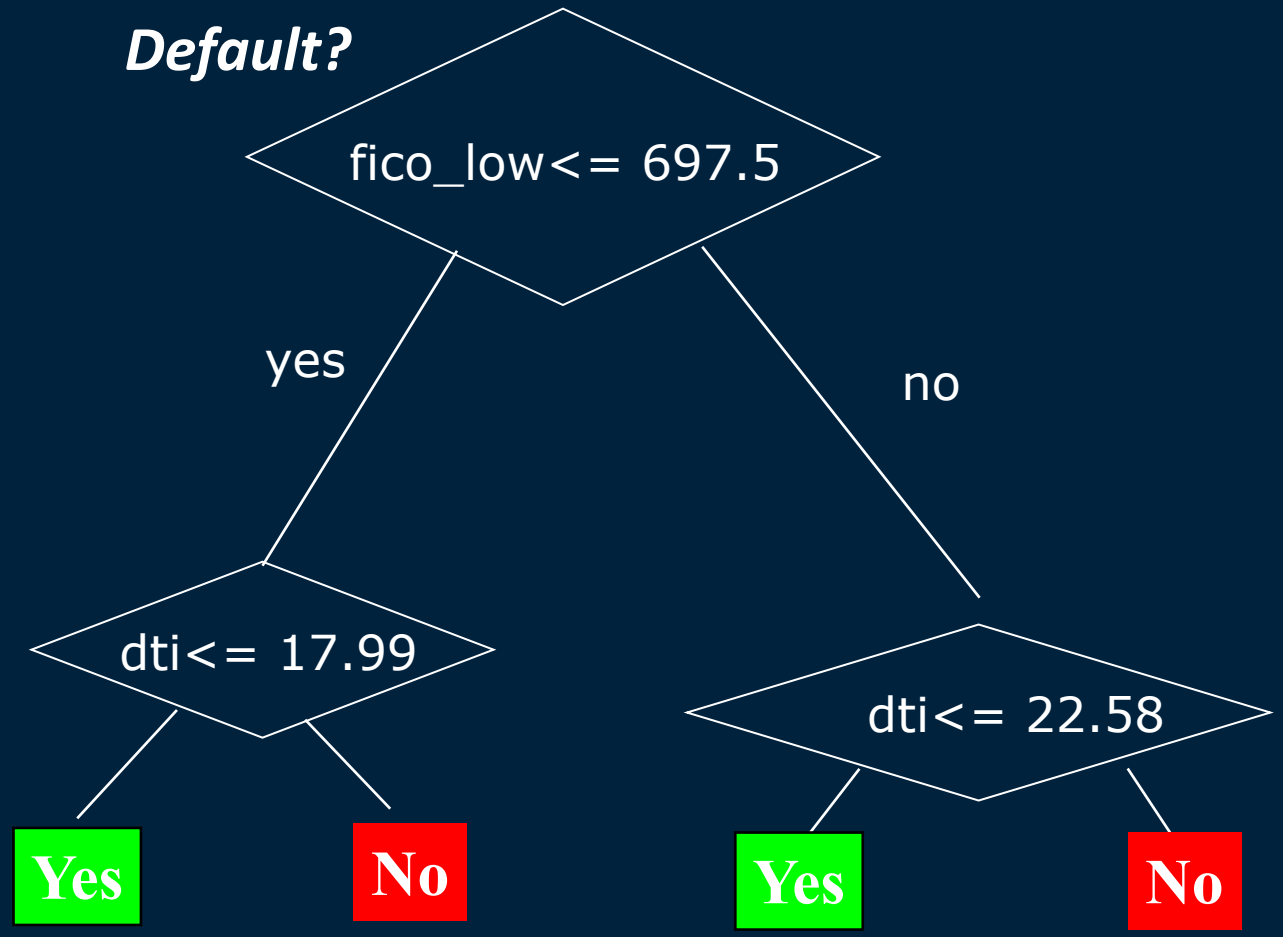
Feature Engineering

Examples

- loan_amnt
- funded_amnt
- annual_inc
- **dti**
- revol_bal
- delinq_2yrs
- pub_rec
- **Fico score**
  - fico_range_high
  - fico_range_low
- revol_util
- **cr_hist**
- **Home_ownership**
  - home_ownership::ANY
  - home_ownership::MORTGAGE
  - home_ownership::OWN
  - home_ownership::RENT

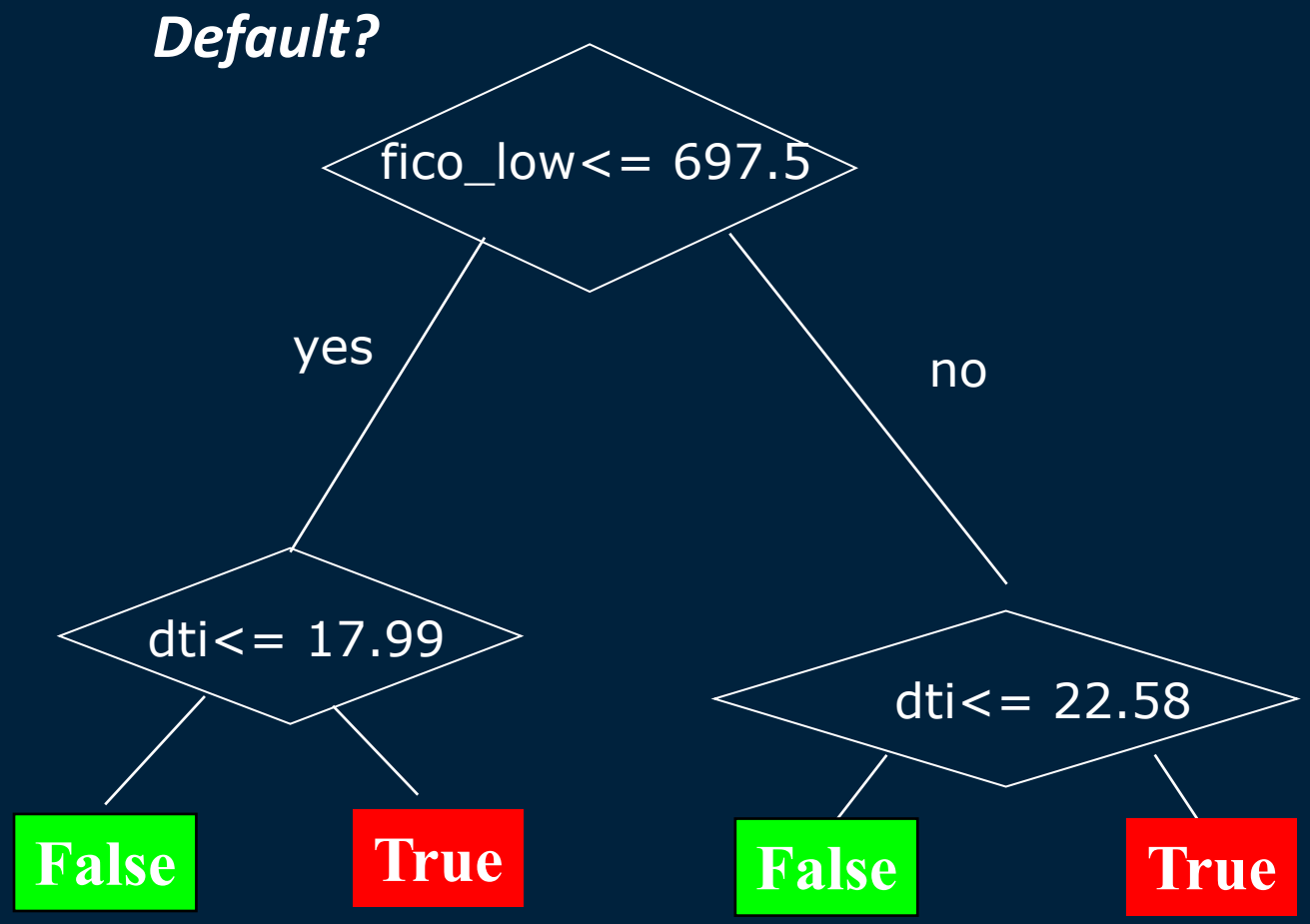
Decision Tree

- Rules for classifying data using attributes
- The tree consists of decision nodes and leaf nodes
- A decision node has two or more branches, each representing values for the attribute tested
- A leaf node attribute produces a classification/decision

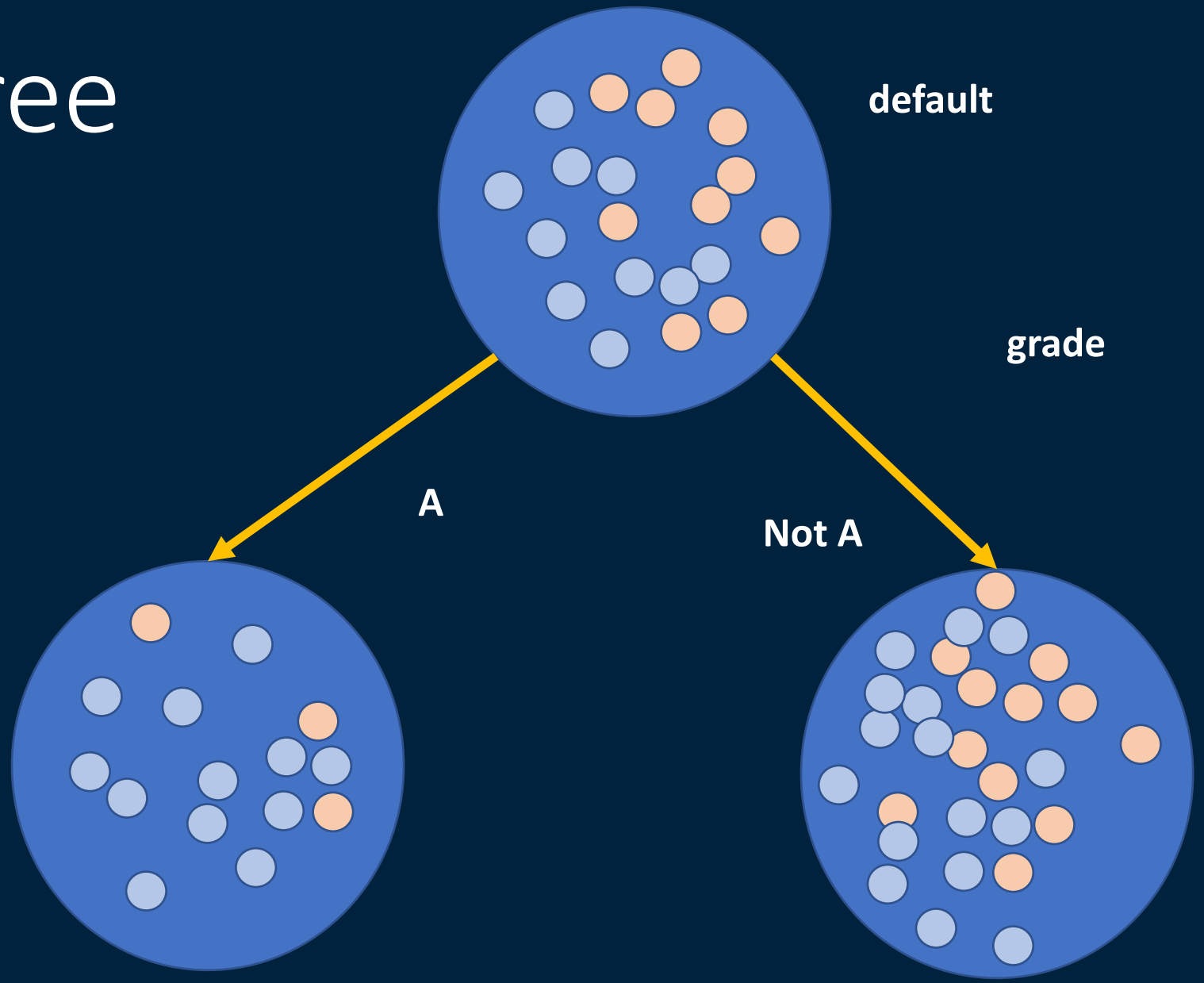


Decision Tree

- The decision tree represents the classification of the table
- It can classify all the objects in the table
- Each internal node represents a test on some property
- Each possible value of that property corresponds to a branch of the tree
- An individual of unknown type may be classified be traversing this tree



Decision Tree



Entropy

- **Measures of impurity**
- **The notion of impurity is closely related to the notion of information**

ID3 Algorithm

- A mathematical algorithm for building the decision tree
- Invented by J. Ross Quinlan in 1979
- Uses Information Theory invented by Shannon in 1948
- Builds the tree from the top down, with no backtracking
- Information Gain is used to select the most useful attribute for classification

Entropy

- A formula to calculate the homogeneity of a sample.
- A completely homogeneous sample has entropy of 0.
- An equally divided sample has entropy of 1.
- Entropy H(X) of a random variable X is defined by

$$H(X) = -\sum p(x) \log p(x)$$

Information Gain

- The information gain is based on the decrease in entropy after a dataset is split on an attribute.
- Which attribute creates the most homogeneous branches?
- First the entropy of the total dataset is calculated.
- The dataset is then split on the different attributes.
- The entropy for each branch is calculated. Then it is added proportionally, to get total entropy for the split.
- The resulting entropy is subtracted from the entropy before the split
- The result is the Information Gain, or decrease in entropy
- The attribute that yields the largest IG is chosen for the decision node
- The ID3 algorithm is run recursively on the non-leaf branches, until no further information gains are possible

Advantages of ID3

- Understandable prediction rules are created from the training data
- Builds the fastest tree
- Builds a short tree
- Only need to test enough attributes until all data is classified or no information gain possible
- Finding leaf nodes enables test data to be pruned, reducing number of tests
- Whole dataset is searched to create tree

Disadvantages of ID3

- Data may be over-fitted or over-classified, if a small sample is tested
- Only one attribute at a time is tested for making a decision
- Classifying continuous data may be computationally expensive, as many trees must be generated to see where to break the continuum

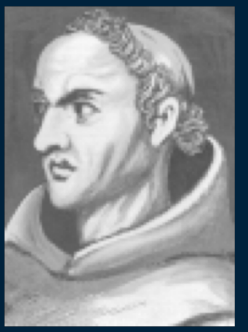
Application of ID3

- In classifying any given instance, the tree does not use all the properties in the table
- Decision tree for credit risk assessment
  - If a person has a good fico score and low debit, we ignore her collateral income and classify her as low risk
  - In spite of omitting certain tests, the tree classifies all examples in the table

Efficiency of ID3

- ID3 algorithm assumes that a good decision tree is the simplest decision tree
- Heuristic:
  - Preferring simplicity and avoiding unnecessary assumptions
  - Known as Occam‘s Razor

Occam Razor



- Occam Razor was first articulated by the medieval logician William of Occam in 1324
    - born in the village of Ockham in Surrey (England) about 1285, believed that he died in a convent in Munich in 1349, a victim of the Black Death
    - It is vain do with more what can be done with less..
- We should always accept the simplest answer that correctly fits our data
- The smallest decision tree that correctly classifies all given examples

sklearn library

In [47]:
```
# Load sklearn utilities
# ----------------------
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

from sklearn.metrics import accuracy_score, classification_report, roc_auc_
from sklearn.calibration import calibration_curve

# Load classifiers
# ----------------
from sklearn.tree import DecisionTreeClassifier
```

sklearn library

In [48]:
```
decision_tree = DecisionTreeClassifier(max_leaf_nodes=16
                                       ,criterion='entropy')
model_name = "Decision tree"
random_state = default_seed
output_to_file = True
print_to_screen = True
```

In [40]:
```
start_time = time.time()
mydecision_tree=decision_tree.fit(X_train, y_train)
end_time = time.time()
print("Fit time: " + str(round(end_time - start_time, 2)) + " seconds")
```

```
Fit time: 0.14 seconds
```

sklearn library

In [48]:
```
decision_tree = DecisionTreeClassifier(max_leaf_nodes=16
                                       ,criterion='entropy')
model_name = "Decision tree"
random_state = default_seed
output_to_file = True
print_to_screen = True
```

In [40]:
```
start_time = time.time()
mydecision_tree=decision_tree.fit(X_train, y_train)
end_time = time.time()
print("Fit time: " + str(round(end_time - start_time, 2)) + " seconds")
```

```
Fit time: 0.14 seconds
```

Tree Representation
sklearn library

In [72]:
```
n_nodes = decision_tree.tree_.node_count
children_left = decision_tree.tree_.children_left
children_right = decision_tree.tree_.children_right
feature = decision_tree.tree_.feature
threshold = decision_tree.tree_.threshold
```

In [44]:
```
print(children_left)
print(children_right)
print(feature)
```

```
[ 1  3  5 11  9  7 17 23 15 13 21 25 19 -1 29 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 27 -1 -1 -1 -1]
[ 2  4  6 12 10  8 18 24 16 14 22 26 20 -1 30 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 28 -1 -1 -1 -1]
[ 9  3  3 19 19 19 14  2  8 34  3  2 11 -2  3 -2 -2 -2 -2 -2 -2 -2 -2 -2
 -2 -2  0 -2 -2 -2 -2]
```

Tree Representation
sklearn library

- The decision estimator has an attribute called ***tree_*** which stores the entire tree structure and allows access to low level attributes
- The binary tree ***tree_*** is represented as a number of parallel arrays The i-th element of each array holds information about the node `i`
- Node 0 is the tree's root
- Some of the arrays only apply to either leaves or split nodes, in these cases the values of nodes of the other type ***are arbitrary***

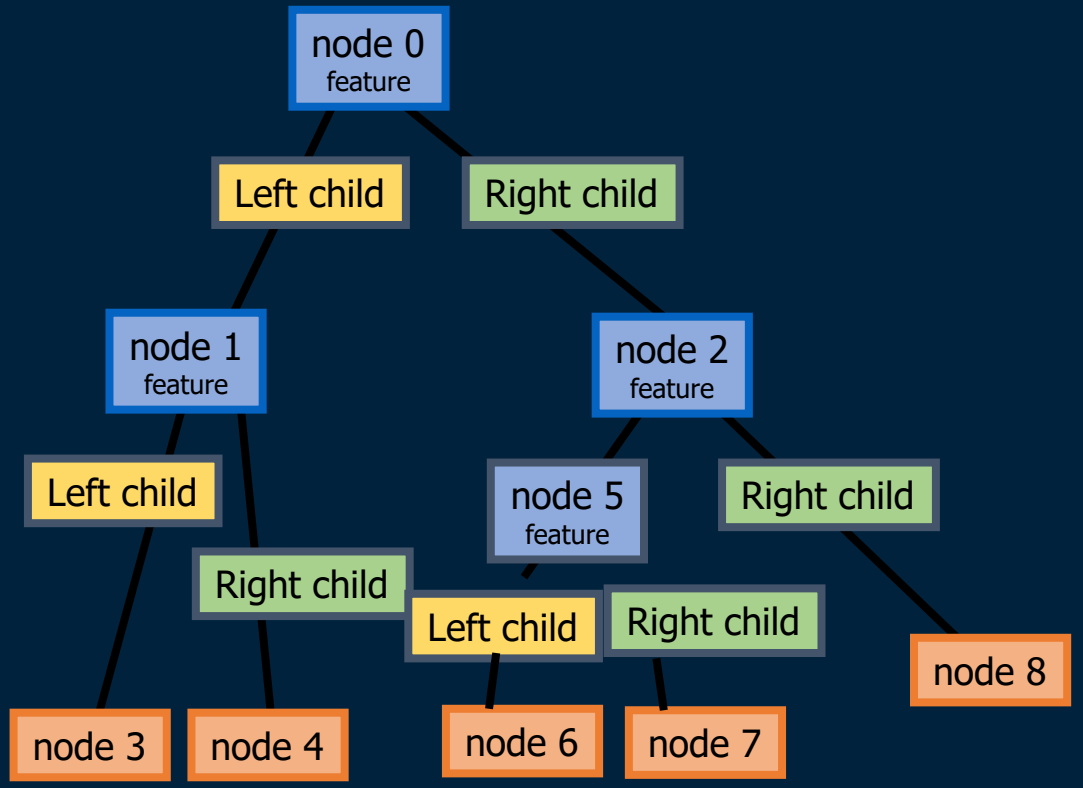
Among those arrays, we have:

- ***left_child***, id of the left child of the node
- ***right_child***, id of the right child of the node
- ***feature***, feature used for splitting the node
- ***threshold***, threshold value at the node

```
In [98]: print(children_left)
         print(children_right)
         print(feature)
         print([float((str(n).split('.'))[0]+'.'\
                      +(str(n).split('.'))[1][:2])\
               for n in list(threshold)])

[ 1  3  5 -1 -1  7 -1 -1 -1]
[ 2  4  6 -1 -1  8 -1 -1 -1]
[ 9  3  3 -2 -2 19 -2 -2 -2]
[697.5, 17.98, 22.19, -2.0, -2.0, 0.5, -2.0, -2.0, -2.0]
```

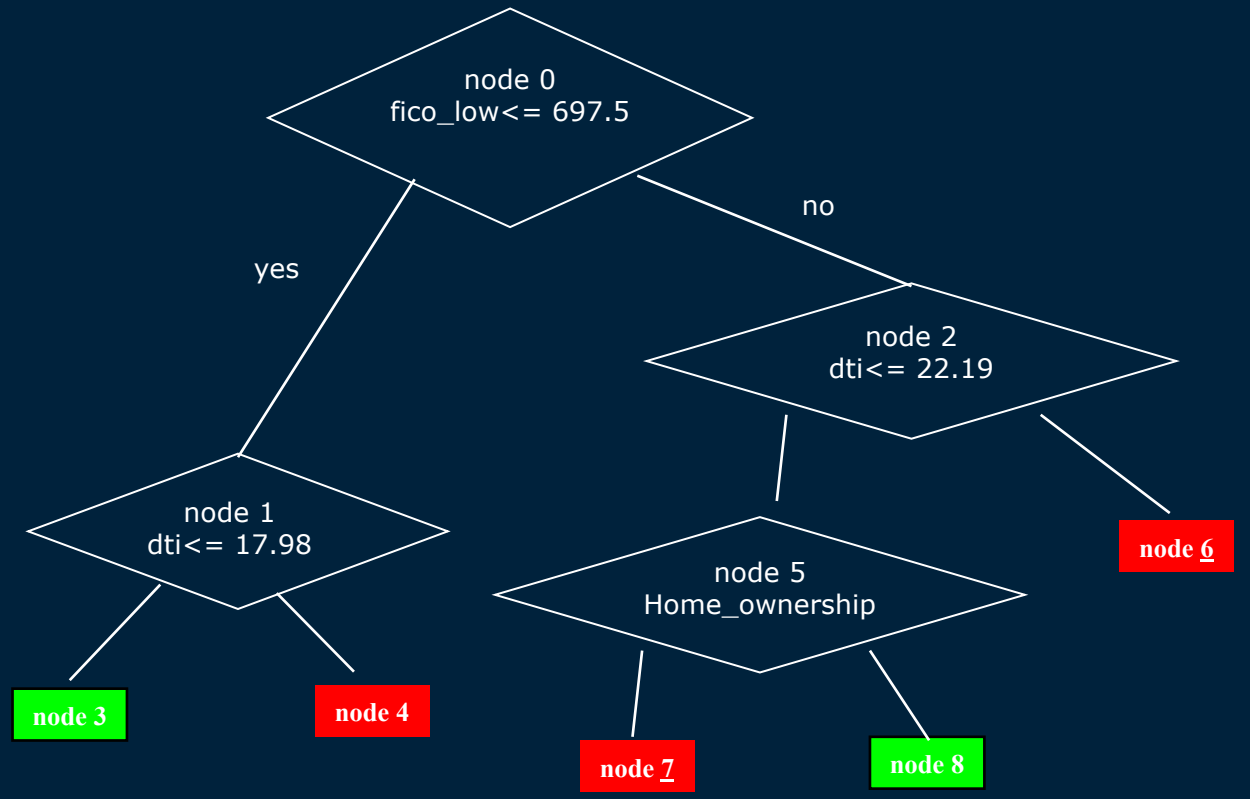
Tree Representation
sklearn library



Tree Representation
sklearn library

```
In [98]: print(children_left)
         print(children_right)
         print(feature)
         print([float((str(n).split('.'))[0]+'.'\
                      +(str(n).split('.'))[1][:2])\
                for n in list(threshold)])
```

```
[ 1  3  5 -1 -1  7 -1 -1 -1]
[ 2  4  6 -1 -1  8 -1 -1 -1]
[ 9  3  3 -2 -2 19 -2 -2 -2]
[697.5, 17.98, 22.19, -2.0, -2.0, 0.5, -2.0, -2.0, -2.0]
```



Traversing a tree
sklearn library

In [86]:
```
# The tree structure can be traversed to compute various properties such
# as the depth of each node and whether or not it is a leaf.
node_depth = np.zeros(shape=n_nodes, dtype=np.int64)
is_leaves = np.zeros(shape=n_nodes, dtype=bool)
stack = [(0, -1)]  # seed is the root node id and its parent depth
while len(stack) > 0:
    node_id, parent_depth = stack.pop()
    node_depth[node_id] = parent_depth + 1

    # If we have a test node
    if (children_left[node_id] != children_right[node_id]):
        stack.append((children_left[node_id], parent_depth + 1))
        stack.append((children_right[node_id], parent_depth + 1))
    else:
        is_leaves[node_id] = True
```

Traversing a tree
sklearn library

In [88]:
```
print("The binary tree structure has %s nodes and has "
      "the following tree structure:"
      % n_nodes)
for i in range(n_nodes):
    if is_leaves[i]:
        print("%snode=%s leaf node." % (node_depth[i] * "\t", i))
    else:
        print("%snode=%s test node: go to node %s if X[:, %s] <= %s else to
              "node %s."
              % (node_depth[i] * "\t",i,children_left[i],
                 selected_features[feature[i]],threshold[i],
                 children_right[i],))
print()
```

Traversing a tree sklearn library

```
The binary tree structure has 9 nodes and has the following tree structur
e:
node=0 test node: go to node 1 if X[:, fico_range_low] <= 697.5 else to n
ode 2.
        node=1 test node: go to node 3 if X[:, dti] <= 17.985000610351562
else to node 4.
        node=2 test node: go to node 5 if X[:, dti] <= 22.19499969482422
else to node 6.
                node=3 leaf node.
                node=4 leaf node.
                node=5 test node: go to node 7 if X[:, home_ownership::MO
RTGAGE] <= 0.5 else to node 8.
                node=6 leaf node.
                        node=7 leaf node.
                        node=8 leaf node.
```